AULA Nº 08
SISTEMAS OPERACIONAIS

Threads

Contextualizando

Na aula passada

Sincronização de Processos

Aula de hoje

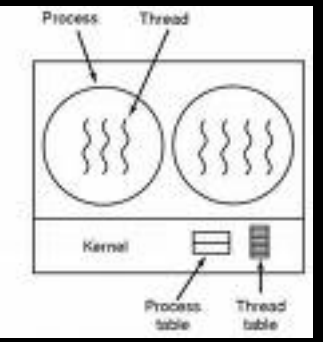
Threads

O Modelo de Processo

1) Utilizado para agrupar recursos
2) Um espaço de endereço (0 até algum endereço máximo do processo) e **uma única linha de execução (Thread)**
3) Agrupamento de recursos (espaço de endereço com texto e dados do programa; arquivos abertos, processos filhos, tratadores de sinais, alarmes pendentes etc)

Modelo da Thread

1) Um espaço de endereço e múltiplas linhas de controle
2) Conjunto de threads compõe as linhas de execuções de um processo
3) Threads compartilham um mesmo espaço de endereço (sendo menos independentes que processos)
4) Possuem recursos particulares (PC, registradores, pilha)



Threads (Vantagens)

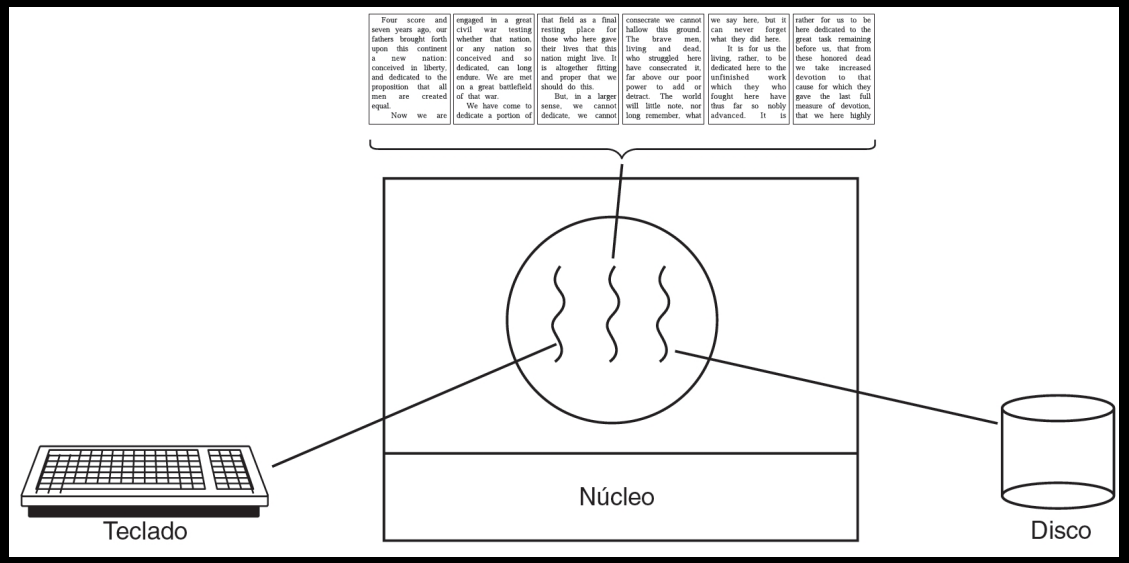
1) Em muitas aplicações há múltiplas atividades ao mesmo tempo
2) Podemos decompô-las em atividades paralelas
3) Algumas tarefas precisam do compartilhamento do espaço de endereçamento
4) CPU-bound e I/O-bound podem se sobrepor, acelerando a aplicação (fica a dica ao programador)

Threads (Vantagens)

1) São mais rápidas de criar e destruir que processos
2) Algumas vezes até 100 vezes mais rápidas
3) Úteis em sistemas com múltiplas CPUs -> paralelismo real

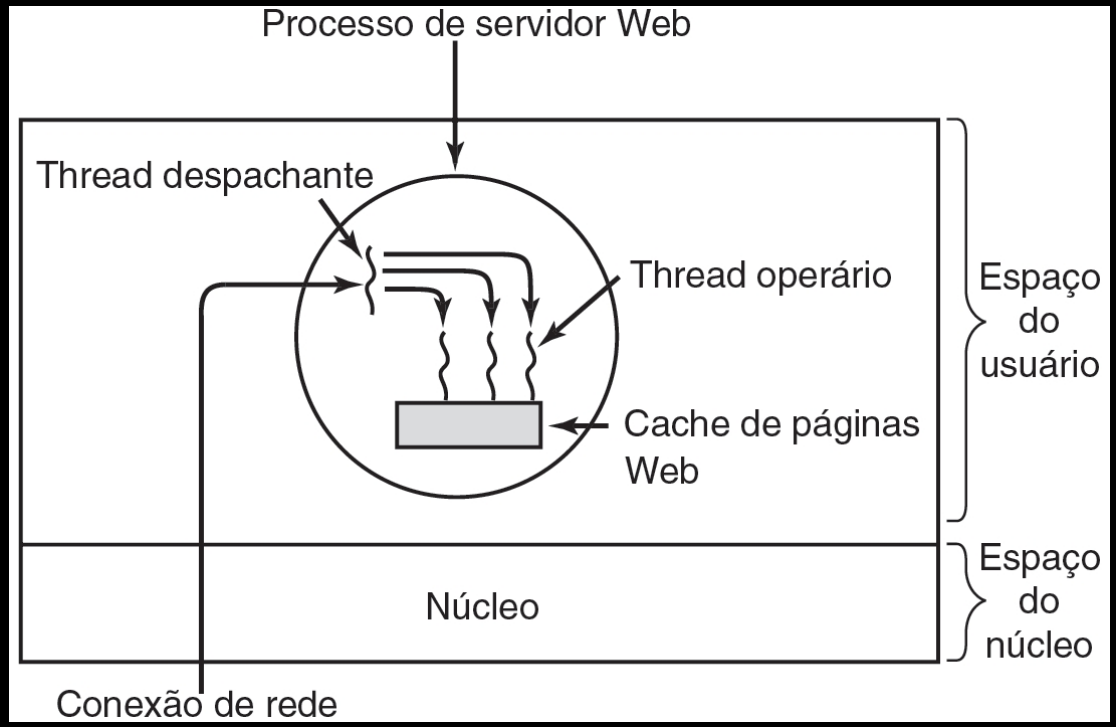
Threads (Exemplos)

1) Processador de texto
2) Processos separados não funcionam - **o documento tem que estar compartilhado**
3) Threads para: Identação, fonte, correção, mudança de linha, etc.



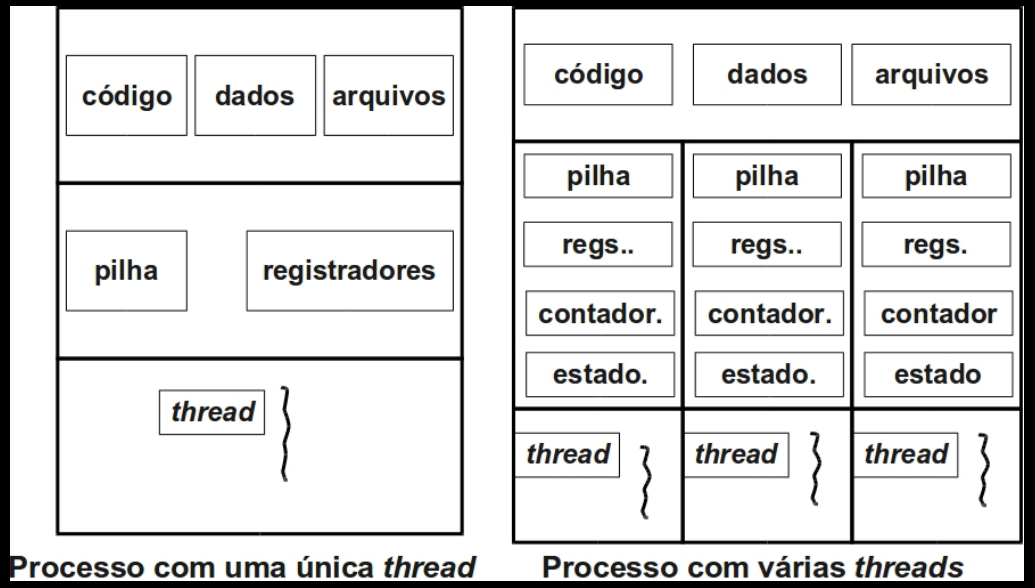
Threads (Exemplo: Servidor Web)

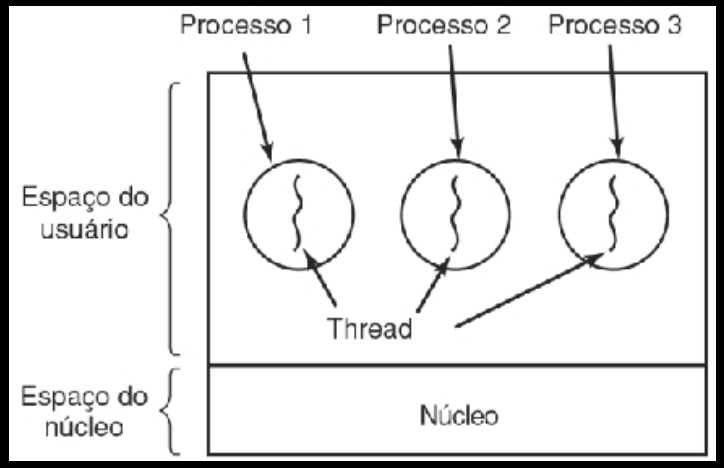
1) O despachante (i) lê as requisições de trabalho que chegam, (ii) escolhe uma thread operário ociosa e (iii) entrega a requisição. A thread operário (iv) lê a cache, caso não encontre a informação, (v) inicializa uma leitura de disco

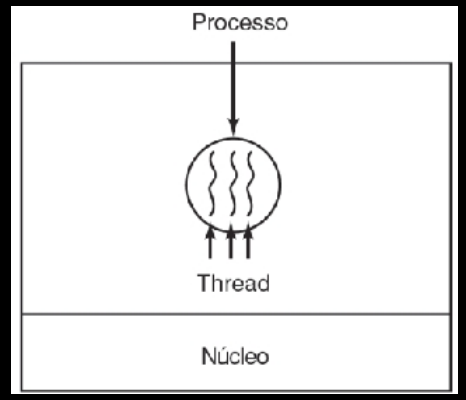


Processos vs. Threads

3 Processos com uma thread cada





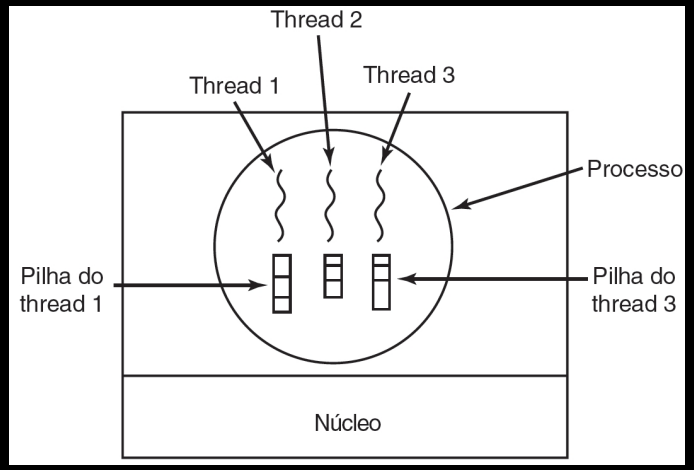


1 Processo

Com três threads

Processos vs. Threads

Cada thread tem sua própria pilha de execução (pois chamam rotinas diferentes), embora compartilhe o espaço de endereçamento e todos seus dados

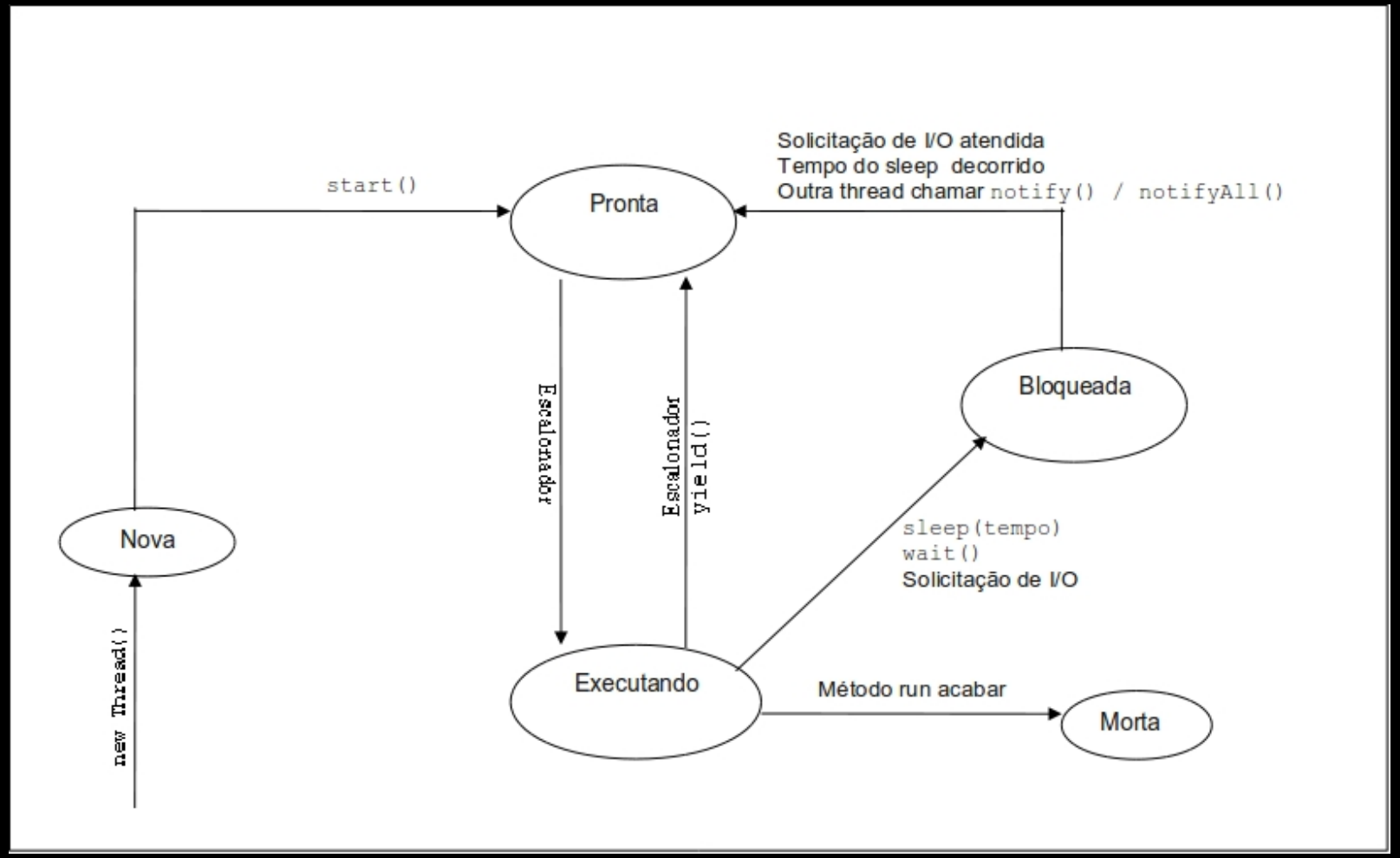


Espaço de endereçamento	Contador de programa
Variáveis globais	Registradores
Arquivos abertos	Pilha
Processos filhos	Estado
Alarmes pendentes	
Sinais e manipuladores de sinais	

Problemas com as Threads

1) Como cada thread pode ter acesso a qualquer endereço de memória dentro do espaço de endereçamento do processo;
   a) uma thread pode ler, escrever ou apagar a pilha ou as variáveis globais de outra thread
   b) Exemplo: a = b + c;
      x = a + y;
2) Necessidade de sincronizar a execução

Estados de Threads



Threads – Implementação com Pacote PTreads do POSIX

Chamada de thread	Descrição
pthread_create	Cria um novo thread
pthread_exit	Conclui a chamada de thread
pthread_join	Espera que um thread específico seja abandonado
pthread_yield	Libera a CPU para que outro thread seja executado
pthread_attr_init	Cria e inicializa uma estrutura de atributos do thread
pthread_attr_destroy	Remove uma estrutura de atributos do thread

Threads – Implementação com Pacote PTread do POSIX

```c
#include <stdio.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
  int pid;
  pid = fork();

  /* Ocorreu um erro */
  if (pid < 0) {
    fprintf( stderr, "Erro ao criar processo" );
  }

  /* Processo filho */
  else if (pid == 0) {
    execlp ("/bin/ls", "ls", NULL);
  }

  /* Processo pai */
  else if (pid > 0) {
    wait (NULL);
    printf ("Processo Pai terminou.\n");
  }
}
```

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

#define NUMERO_DE_THREADS 10

void *ola_mundo(void *tid) {
  printf("Ola mundo. Saudacoes da thread %d\n",tid);
  pthread_exit(NULL);
}

int main(int argc, char *argv[]) {
  pthread_t threads[NUMERO_DE_THREADS];
  int status, i;

  for (i=0; i<NUMERO_DE_THREADS; i++) {
    printf("Thread principal. Criando thread %d\n",i);
    status = pthread_create(&threads[i], NULL,
                            ola_mundo, (void *)i);
    if (status != 0) {
      printf("pthread_create retornou o codigo de erro
                                          %d\n",status);
      exit(-1);
    }
  }
  exit(0);
}
```

Threads – Implementação em Java

Estendendo a **class Thread**

Criar thread: Instanciar classe que herda da classe `Thread'

class Thread possui todo o código para criar e executar threads

```
class ThreadSimples extends Thread {

    public void run() {
        System.out.println("Ola de uma
                            nova thread!");
    }

    public static void main(String
                                args[]) {
     Thread thread =
                  new ThreadSimples();
     thread.start();

        System.out.println("Ola da
                        thread original!");
    }
}
```

Threads – Implementação em Java

Joining a Thread

1) Permite a uma thread esperar que outra termine
2) A thread principal esperará thread2 morrer

```
class ThreadSimples extends Thread {

   public void run() {
      System.out.println("Ola de uma nova thread! "
                         + super.toString() + ".");
   }

   public static void main(String args[]) {
      ThreadSimples thread = new ThreadSimples();
      ThreadSimples thread2 = new ThreadSimples();

      thread.start();
      thread2.start();

      try {
         thread2.join();
      }
      catch (InterruptedException e) {
         e.printStackTrace();
      }

      System.out.println("Ola da thread original!");
   }
}
```

Threads – Implementação em Java

Sleeping a Thread

1) A thread atual fica bloqueada por um número de milisegundos
2) Precisa capturar InterruptedException

```
try {
    Thread.sleep(1);
}
catch (InterruptedException e) {
    e.printStackTrace();
}
```

Tipos de Threads

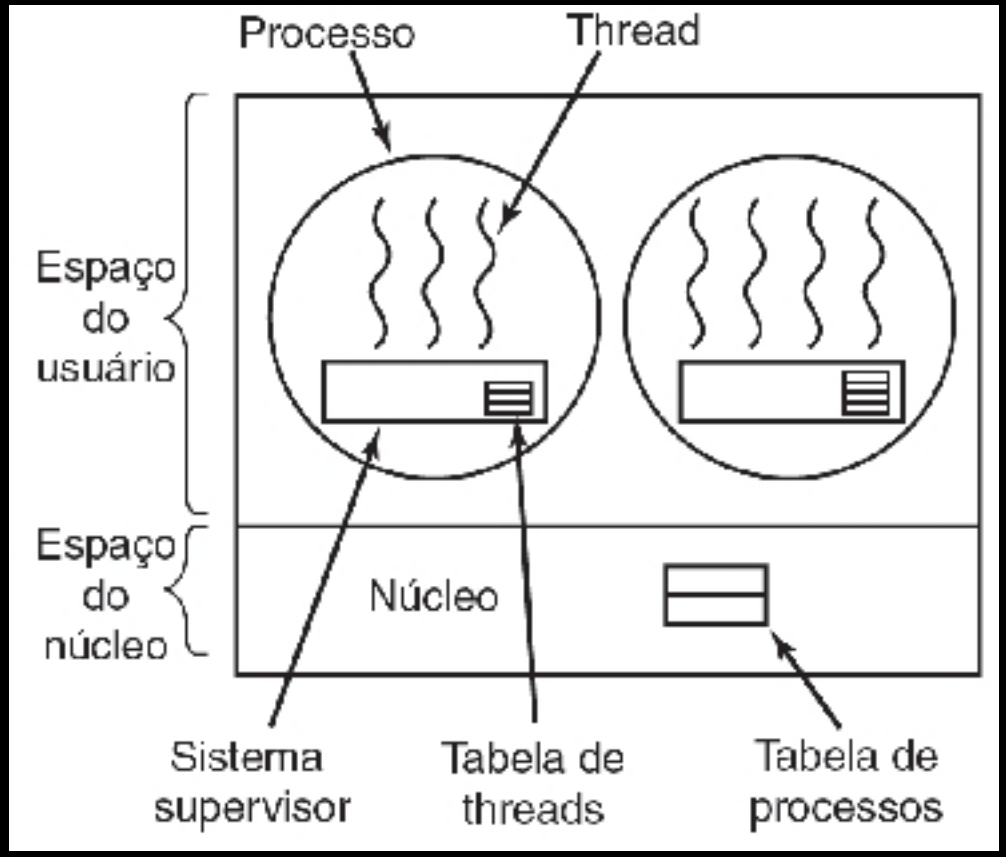
1) No Modo Usuário
2) No Modo Núcleo
3) Híbrido

Threads no Modo Usuário

1) Implementadas totalmente no espaço do usuário
2) Por meio de uma biblioteca (criação, exclusão, execução etc, não necessariamente gerenciamento)
3) Criação e escalonamento são realizados sem o conhecimento do kernel
4) Para o kernel, é como se rodasse um programa monothread
5) Gerenciadas como processos no Kernel

Threads no Modo Usuário

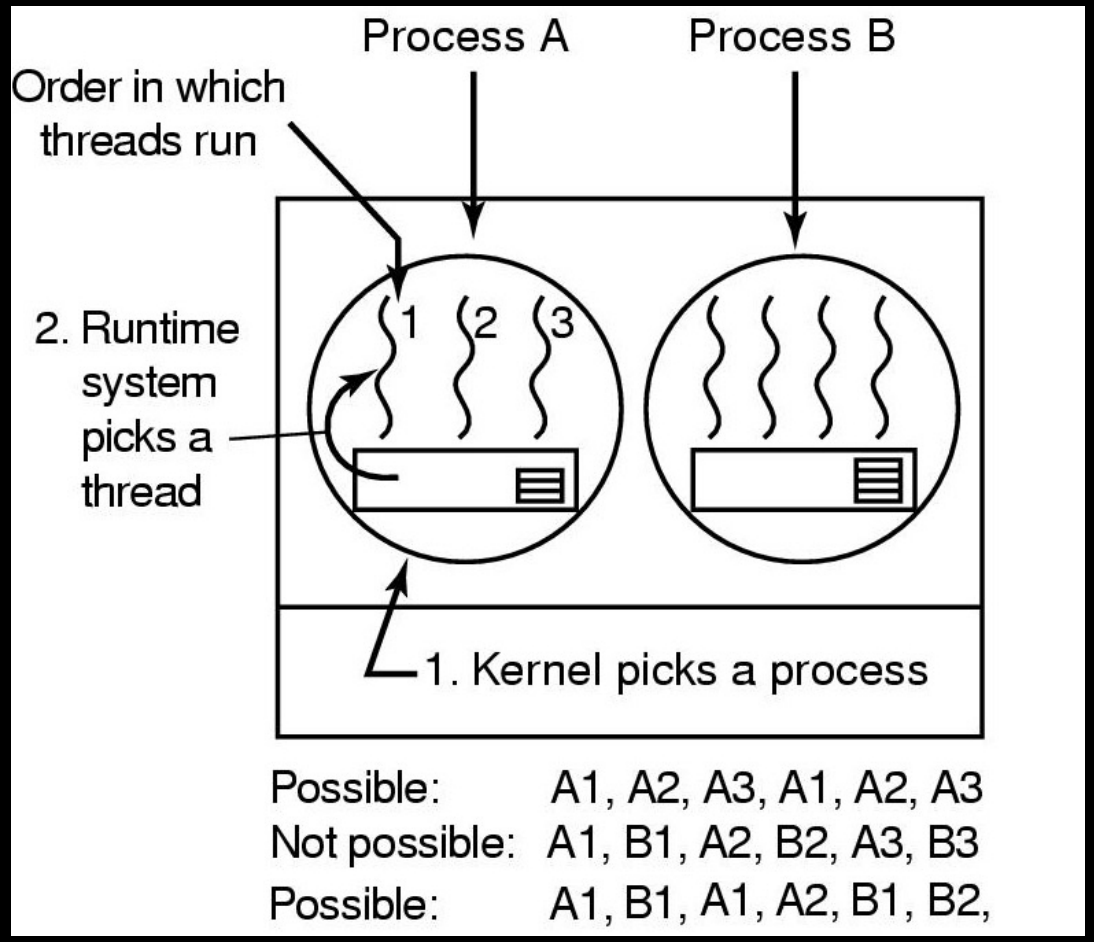
1) Cada processo possui **sua própria tabela de threads**
2) Como uma tabela de processos, gerenciada pelo **runtime**
3) Controla apenas as propriedades da thread (PC, ponteiro da pilha, registradores, estado etc)



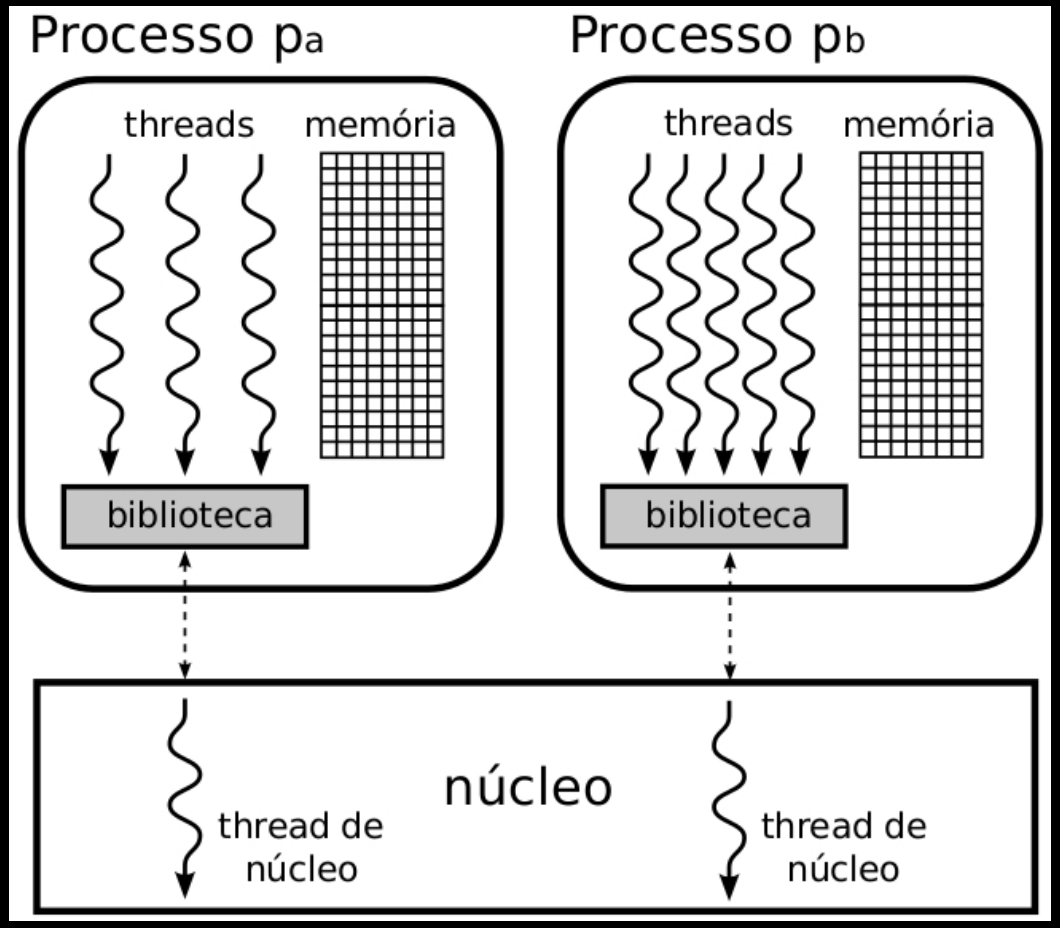
Escalonamento-Thread Usuário

1) O núcleo escolhe um processo e passa o controle a ele que **escolhe uma thread**

2) A gerência da thread fica no espaço do usuário e o núcleo só escalona em nível de processo



Modelo N:1


Processo pa
Processo pb
threads
memória
threads
memória
biblioteca
biblioteca
núcleo
thread de núcleo
thread de núcleo

Threads no Modo Núcleo

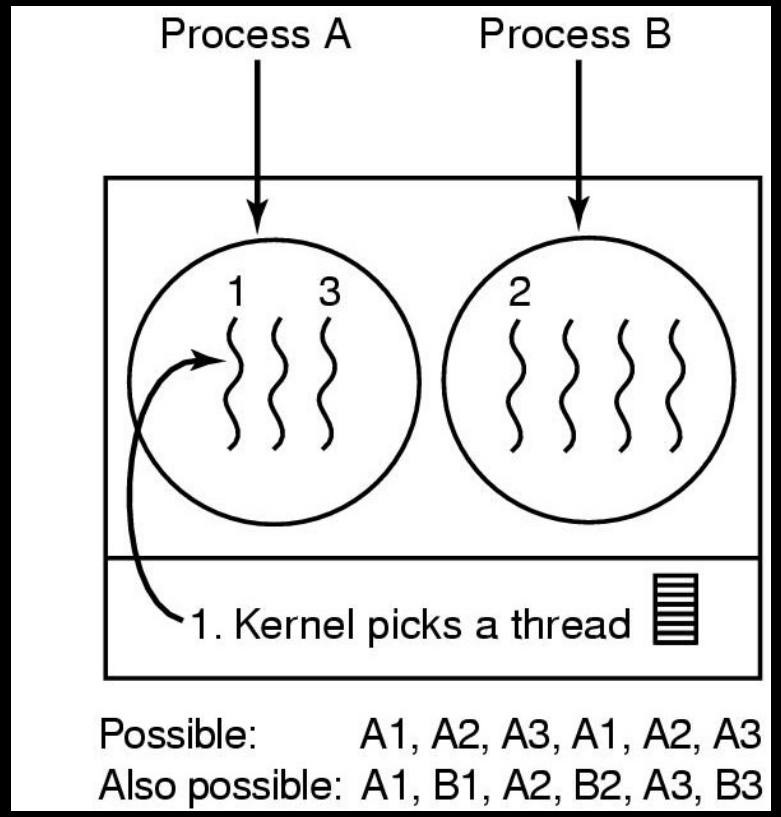
1) Suportadas diretamente pelo SO
2) Criação, escalonamento e gerenciamento são feitos pelo kernel
3) O núcleo possui tabela de threads (com todas as threads do sistema) e tabela de processos separadas
4) As tabelas de threads possuem as mesmas só que agora estão implementadas no kernel
5) Os algoritmos mais usados são Round Robin e Prioridade

Threads no Modo Núcleo

1) Agora, as tabelas de threads estão no núcleo
2) Gerenciar threads em modo kernel é mais caro devido à alternância entre modo usuário e modo kernel
3) Mudança de contexto pode ser envolvido na mudança de threads
4) Criar e destruir threads no núcleo é mais caro
5) Exemplo: Linux, Família Windows, OS/2, Solaris 9 (mapeia 1 thread usuário para 1 de kernel, i.e. 1:1)

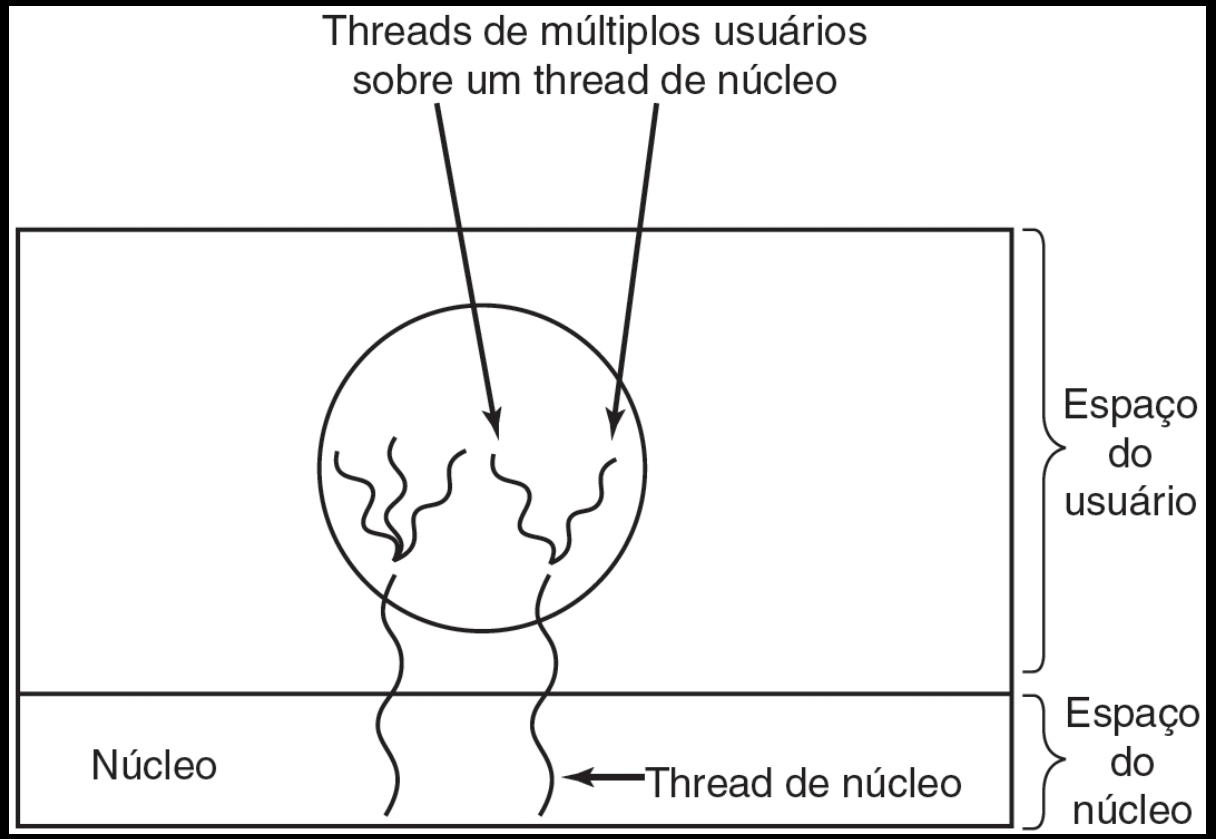
Escalonamento Threads Núcleo

1) O núcleo escolhe a thread diretamente
2) A thread é quem recebe o quantum, sendo suspensa se excedê-lo
3) Thread bloqueada por E/S não bloqueia o processo
4) Permite múltiplas threads em paralelo



Threads Híbridas

1) Seguem o modelo N para M:
2) N threads de usuário são mapeadas em M <= N threads de núcleo
3) Ex.: Solaris até versão 8, HP-UX, Tru64 Unix



Concluindo

Foram abordados nesta aula:

- **Threads**

Concluímos o Capítulo 2 de Sistemas Operacionais Modernos; Tanenbaum, A. S. 4ª Edição